

20 LECCIONES DE BASIC
PARA APRENDER CON EL SPECTRUM







Spectrum
16K/48K/PLUS

VIDEO BASIC
Una publicación de
INGELEK JACKSON
Director editor por INGELEK:
Antonio M. Ferrer
Director editor por JACKSON HISPANIA:
Lorenzo Bertagnolio
Director de producción:
Vicente Robles
Autor: Softidea
Redacción software italiano:
Francesço Franceschini,
Stefano Cremonesi
Redacción software castellano:
Fernando López, Antonio Carvajal,
Alberto Caffarato, Pilar Manzanera
Diseño gráfico:
Studio Nuovaidea
Ilustraciones:
Cinzia Ferrari, Silvano Scolari,
Equipo Galata
Ediciones INGELEK, S. A.
Dirección, redacción y administración,
números atrasados y suscripciones:
Avda. Alfonso XIII, 141
28016 Madrid. Tel. 2505820
Fotocomposición: Espacio y Punto, S. A.
Imprime: Gráficas Reunidas, S. A.

ISBN del tomo 4: 84-85831-20-9
ISBN del fascículo: 84-85831-11-X
ISBN de la obra completa: 84-85831-10-1
Depósito Legal: M-15076-1985
Plan general de la obra:
20 fascículos y 20 casetes, de aparición quincenal, coleccionables en 5 estuches.
Distribución en España:
COEDIS, S. A.
Valencia, 245. 08007 Barcelona.
INGELEK JACKSON garantiza la publicación de todos los fascículos y casetes que componen esta obra y el suministro de cualquier número atrasado o estuche mientras dure la publicación y hasta un año después de terminada.
El editor se reserva el derecho de modificar el precio de venta del fascículo, en el transcurso de la obra, si las circunstancias del mercado así lo exigen.
Diciembre, 1985
Impreso en España.

INGELEK



JACKSON

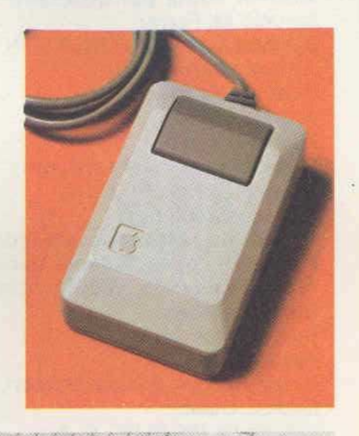
SUMARIO

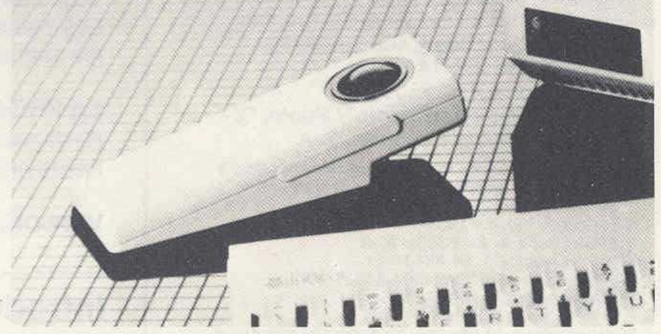
Introducción

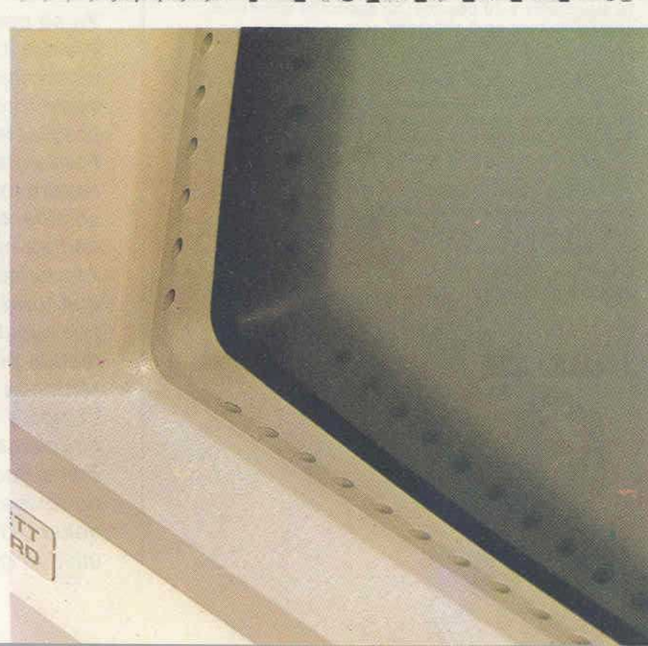
Ya se puede decir que teneis a vuestra disposición todos los elementos necesarios para convertiros en buenos programadores en BASIC.
Pero para conseguir del ordenador resultados aún más sorprendentes solamente queda un camino, el código máquina.
Afortunadamente, no es necesario que todo lo tengamos que programar por nuestra cuenta; muchas veces resulta preferible, hablando en términos de posibilidades reales: tiempo y dinero, comprar directamente software estándar.
Trataremos pues de los procesadores de textos, hojas de cálculo, bases de datos y demás «herramientas» de utilidad general.

Los modernos dispositivos de entrada

Como bien sabemos, el teclado constituye para cualquier ordenador un dispositivo de entrada extremadamente importante, o mejor dicho, fundamental. Sin embargo, existen otros muchos periféricos de entrada que permiten proporcionar al ordenador de una manera sencilla e inmediata todos aquellos datos que se desee introducir en su memoria.
Desde que el tratamiento gráfico ha entrado a formar parte del campo habitual de prestaciones de los ordenadores personales, se han desarrollado nuevos dispositivos para conseguir hacer más fácil e inmediato el coloquio entre hombre y máquina.
Los más difundidos son el ratón, el trackball y la pantalla táctil.





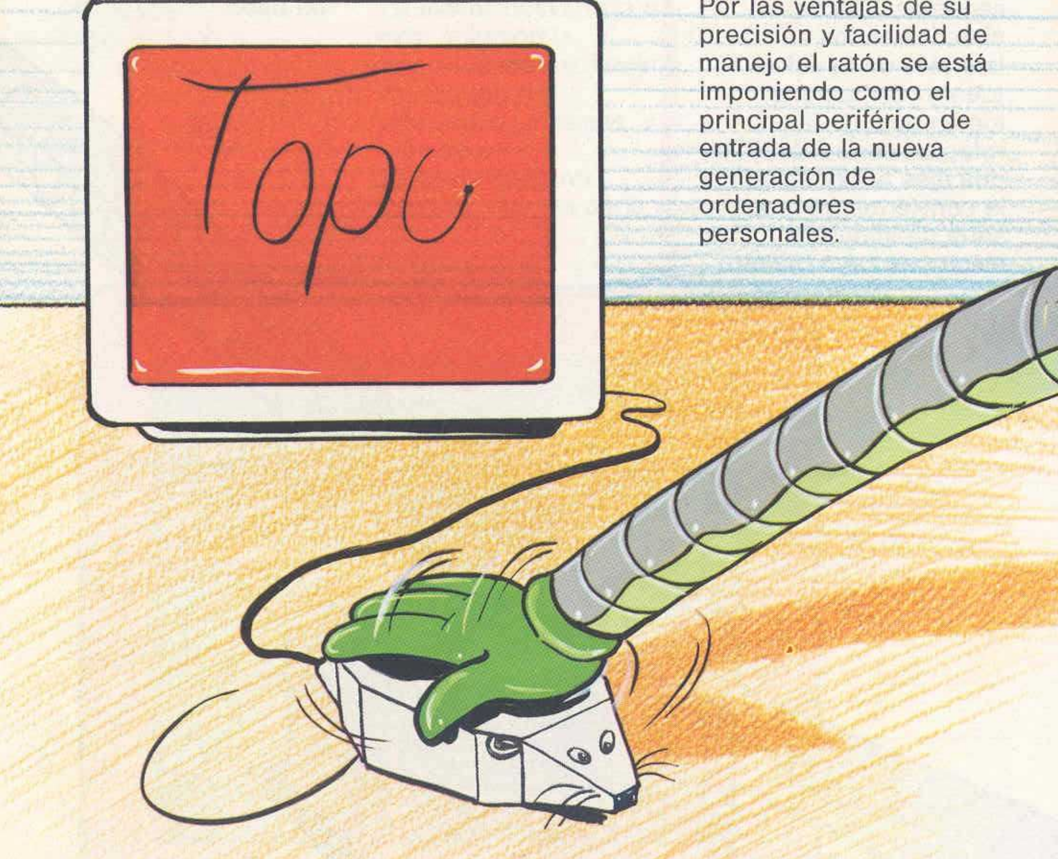


RATON

El ratón (mouse en inglés) viene a ser un trackball boca abajo. Aquí la esfera se hace rotar desplazando la totalidad del dispositivo sobre un plano liso (por ejemplo, una mesa).

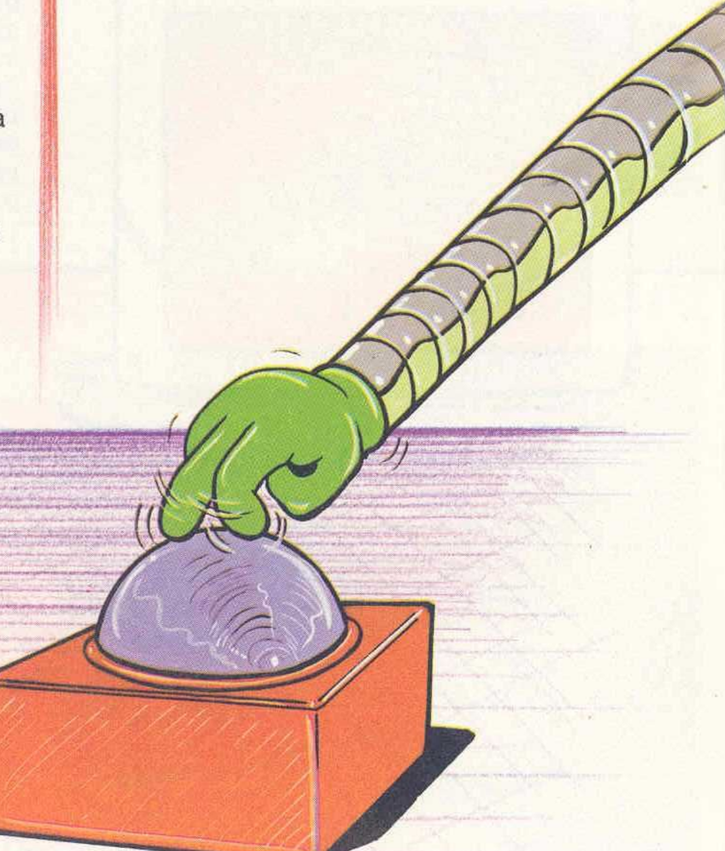
Con respecto al trackball, el mouse necesita un espacio mayor, siendo necesaria una cierta libertad de movimientos para poder maniobrarlo; en compensación la operación resulta mucho más intuitiva, puesto que el desplazamiento del cursor sobre la pantalla recalca con toda fidelidad el que va realizando la mano. Además es posible mover el cursor manteniendo pulsada al mismo tiempo la tecla (o teclas) del ratón, lo que es imposible con el trackball.

Por las ventajas de su precisión y facilidad de manejo el ratón se está imponiendo como el principal periférico de entrada de la nueva generación de ordenadores personales.



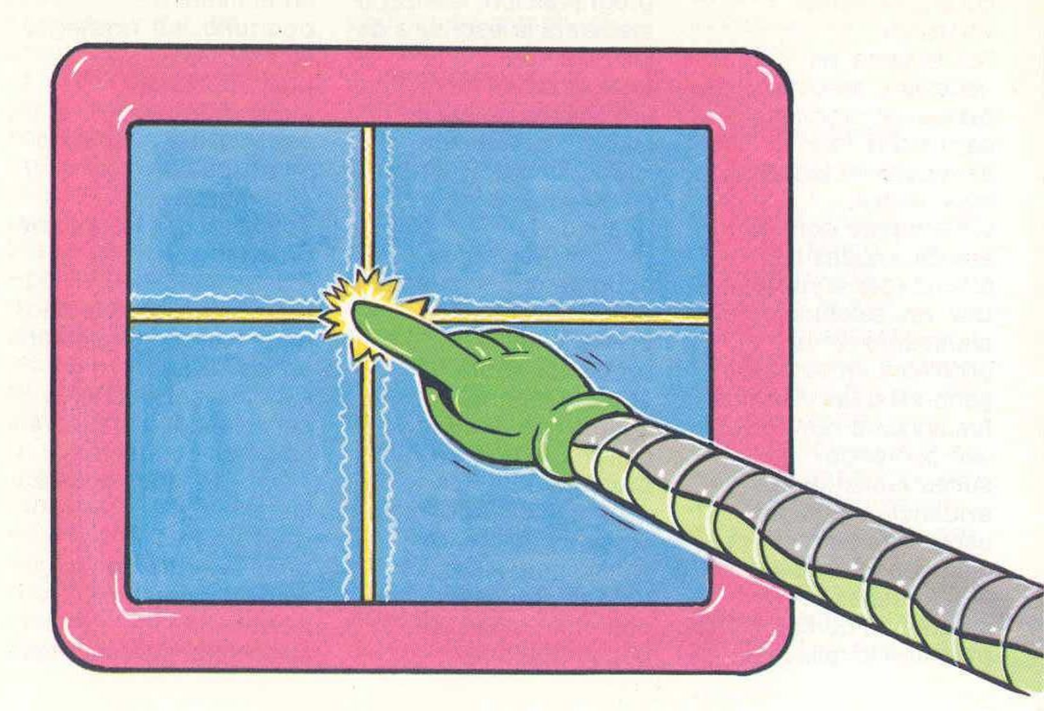
EL TRACKBALL

El trackball (literalmente, «esfera trazadora») es un periférico constituido por una bola completamente lisa que puede girar libremente sobre un soporte fijo y que se pone en movimiento con la palma de la mano. Dos ruedas dentadas con sensores ópticos situadas en los laterales de la esfera (dentro del soporte) toman nota de la rotación con respecto a dos ejes ortogonales y la convierten en una serie de impulsos eléctricos cuyo número es proporcional al ángulo de rotación. Puesto que contrariamente a lo que ocurre con joysticks y paddles no existe un tope final, es posible obtener toda la precisión que se desee, aunque a veces esta ventaja va en detrimento de la velocidad de manejo.
Quedan para el software las tareas de «pilotaje» y el elegir la mejor relación de desmultiplicación, evaluada siempre como un compromiso entre la exactitud y la velocidad de uso.



PANTALLA TACTIL

La idea de la pantalla táctil es muy semejante a la del lápiz óptico, el término inglés «touch-screen» no significa más que pantalla sensible al tacto.
En realidad no es la pantalla la que es sensible al toque del operador, sino una hoja transparente que cubre la pantalla. Un retículo de sensores ópticos hace que esta hoja sea sensible a los dedos o a una pluma.
La mayor desventaja de esta solución la constituye la escasísima resolución del dispositivo. Además, y esto es un inconveniente no despreciable, la capa de grasa que siempre existe sobre la piel termina acumulándose sobre la pantalla, perjudicando la lectura. Por el contrario, el uso de este periférico es extremadamente natural e intuitivo, pudiendo por tanto ser empleado por personas no demasiado expertas.



Software auxiliar

Cualquier ordenador, por más sofisticado y perfeccionado que sea, es incapaz de operar por su cuenta: siempre es necesario emparejar a las características de hardware de la máquina el software apropiado a la resolución del problema o problemas que se pretendan resolver.
Sin embargo, para conseguir este objetivo, el programador tiene que realizar un conjunto de pasos y operaciones que son constantemente variables.
Por lo tanto, es necesario analizar y evaluar el problema bajo todos sus aspectos, incluso los más nimios, examinando con atención todas las dificultades a superar.
Una vez establecido el planteamiento del problema en términos generales, las distintas funciones a realizar se van poniendo sucesivamente en evidencia mediante esquemas adecuados (llamados normalmente —ya lo sabes— diagramas de flujo), que permiten localizar la sucesión lógica de acciones, con las posibles situaciones alternativas y los eventuales momentos de decisión. También en esta fase hay que decidir si la totalidad del conjunto de funciones a realizar se concentrará en un único programa o bien se subdividirá en varios programas y sus correspondientes subprogramas.
Una vez establecidos los diagramas de flujo, el programa pasa finalmente a la verdadera fase de programación, realizada mediante la escritura de las diversas instrucciones en un lenguaje que —como el BASIC— pueda ser «comprendido» por la máquina.
Acabada la programación nos ocuparemos (después de haber grabado previamente el programa sobre un soporte magnético seguro) de poner en marcha la fase de prueba.
Ahora es cuando llega el momento más estimulante y difícil de todo el proceso, cuando aparecen todas las pegas que nos demuestran las posibles lagunas que un análisis apresurado del problema pueda haber dejado.
En esta operación pueden ser de ayuda al programador aquellos conjuntos de programas normalmente llamados «software de ayuda», que le permiten inspeccionar, evaluar y corregir los errores de una forma mucho más rápida y fácil.
Los programas de ayuda más comunes se ocupan de operaciones que a primera vista parecen muy triviales; en el momento oportuno, sin embargo, se convierten en prácticamente indispensables, especialmente cuando se desea aprovechar al máximo las posibilidades del propio ordenador.
Existen muchos programas de este tipo en el mercado; también en las revistas especializadas suelen ser tomados en cuenta.
Los más completos TOOL-KIT (tool significa literalmente «herramienta», precisamente porque estos programas pueden ser considerados como las

herramientas del programador) permiten desarrollar numerosas operaciones, de las que destacaremos:

- **numerar** automáticamente las líneas del programa mientras se escribe. Así se evita tener que escribir todas las veces los números de línea; naturalmente es posible especificar el paso, es decir, el incremento, que se desea incluir entre una y otra línea;
- **renumerar** un programa que haya sufrido muchos ajustes. Esta es una de las posibilidades más apreciadas por los programadores, que si no existieran estos programas a veces ya no podrían insertar nuevas instrucciones. Queda claro que la renumeración de las líneas tiene que implicar —por lo menos en un programa profesional— también la renumeración correcta de las distintas instrucciones GOTO y GOSUB;
- **borrar** bloques de programa. Muchos ordenadores permiten realizar esta acción: desde el sistema operativo de tu Spectrum, en cambio, sólo es posible borrar una línea a la vez. Con un programa de «delete» (borrado) es posible indicar instrucciones del tipo "delete 10-130" (borra todas la líneas entre la 10 y la 130);
- **buscar** una determinada cadena de caracteres dentro del programa. De esta forma se evita tener que buscar trabajosamente la cadena (por ejemplo, el nombre de una variable) a lo largo de todo el listado;
- **sustituir** automáticamente una cadena de caracteres por otra;
- **visualizar** los números de línea del programa según se vayan ejecutando las instrucciones. También esta posibilidad es de gran interés y utilidad; es muy frecuente que el programa funcione correctamente, es decir, que proporcione resultados de salida, pero que estos no se correspondan absolutamente con los que necesitaríamos que aparecieran. Estos son los errores más difíciles de localizar, puesto que no afectan a la sintaxis del programa sino a su lógica. Con un programa de «trace» (traza) se hace posible seguir, a través de la visualización progresiva de los números de línea que se van realizando, el desarrollo del programa en tiempo real, es decir, en el mismo momento en que va ocurriendo cada cosa. Muchas veces es suficiente una ojeada a estos números para ser capaz de resolver un problema provocado por un salto a una línea errónea o por una sucesión de instrucciones no realizada.

Aplicaciones comerciales

Igual que ocurre con la ropa, los programas también pueden hacerse a medida o preconfeccionarse. Como es natural, estos últimos cuestan menos. Los programas proyectados y escritos específicamente de cara a la resolución de un determinado problema son aquellos que desempeñan esta tarea con la máxima velocidad, con el mejor

aprovechamiento de las características técnicas del ordenador empleado, y con la máxima adecuación a las exigencias de quien tenga que usarlo.
Las ventajas de estos programas se pueden resumir en unos pocos pero muy importantes puntos:
– se realizan por profesionales y están garantizados;
– están disponibles en muy poco tiempo (es decir, no requieren largos plazos de espera);
– tienen un precio bastante razonable.
Analizando también las posibles desventajas, la principal es que no pueden satisfacer al cien por cien las exigencias del usuario. Siempre queda un cierto margen de «insatisfacción» debido al hecho de que un programa proyectado para resolver un problema en su conjunto no puede tener en cuenta todos los casos particulares posibles.
Por ejemplo, un programa de contabilidad nunca podrá ser idéntico para un artesano o para una empresa mediana: por esta razón muchos programas están «abiertos», es decir, permiten un margen más o menos amplio

OPERARIO



de modificación para eventuales «personalizaciones». En cambio, hay exigencias auténticamente comunes a un número muy grande de usuarios, como la escritura y el archivo de datos.
Así pues, los más importantes «paquetes de aplicaciones» disponibles en la actualidad son: los tratamientos de texto, las hojas de cálculo y las bases de datos.

Tratamiento de textos

Un programa para el procesamiento de textos (del inglés «Word processor») es una de las posibles aplicaciones que es capaz por sí sola de justificar la compra de un ordenador personal. La característica principal de un tratamiento de textos es la de transformar al ordenador en una máquina de escribir muy especial. Permite escribir a la primera, y desordenadamente si se quiere, puesto que se tiene la posibilidad de modificar cualquier parte del texto sin tener que volverlo a escribir

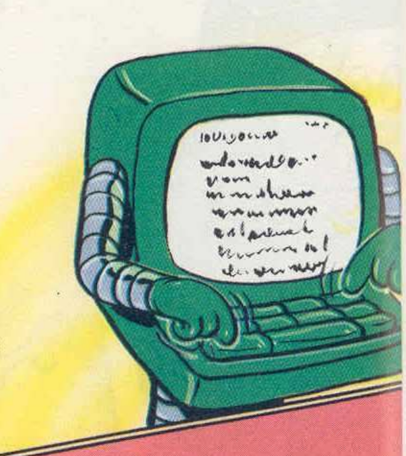




todo desde el principio, para sacar la copia en limpio, y permite obtener con la misma calidad de impresión un número ilimitado de copias.
Su uso suprime de forma definitiva gomas de borrar, artificios correctores, papel carbón, y demás incomodidades.
Además un sistema de tratamiento de textos maneja automáticamente el alineado de las palabras, tanto en el interior de las líneas como en los márgenes, que a su vez pueden modificarse a placer con pocas y elementales instrucciones.
Pero la característica fundamental de un tratamiento de textos es consentir la corrección «dinámica» de todo aquello que se ha escrito: se puede «volver atrás» y corregir en pantalla, dejando al ordenador y a la impresora la tarea de reescribir todo el texto sobre el papel.

Las funciones más importantes y necesarias de un sistema para el tratamiento de textos son:

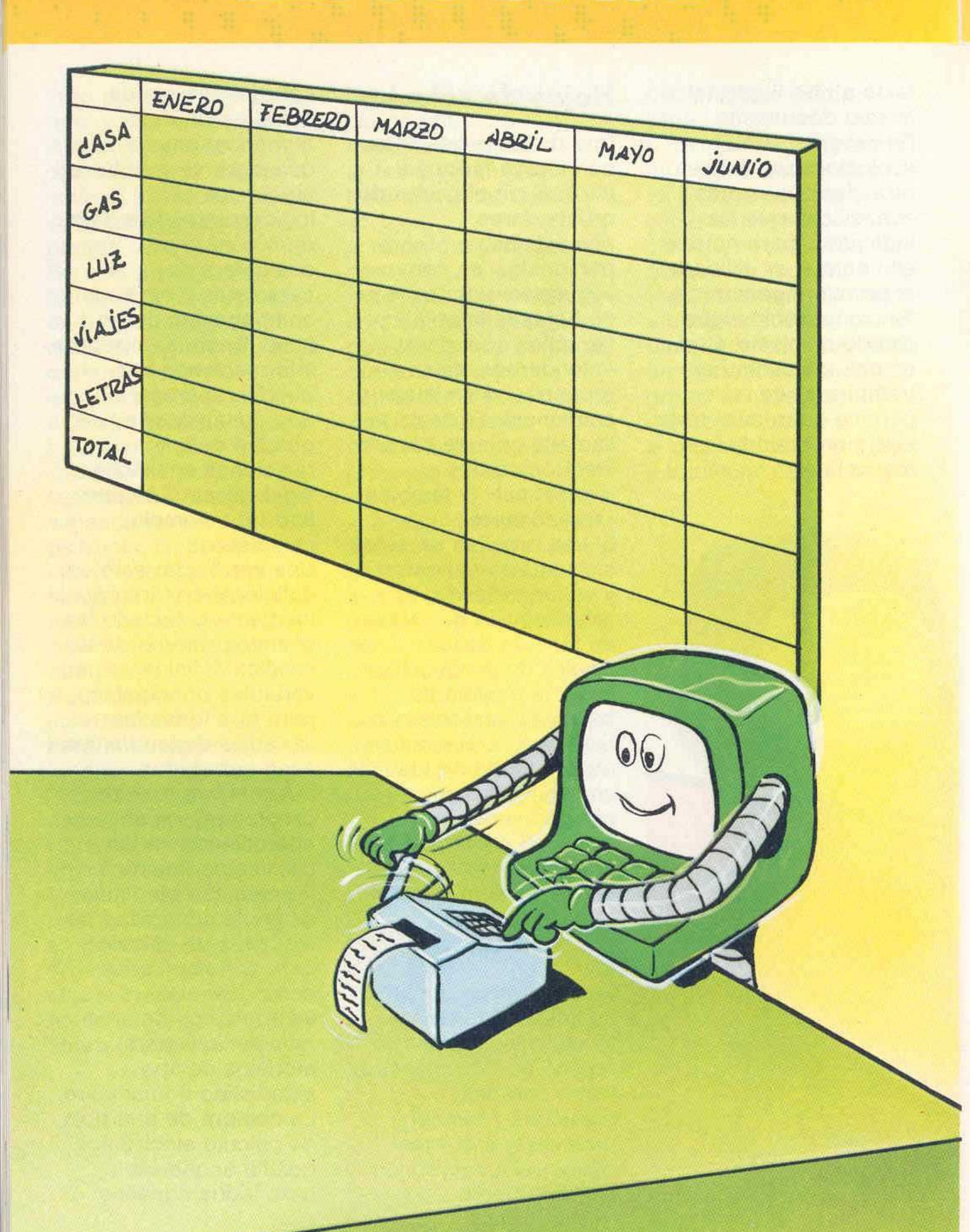
- retorno automático de carro. Así no es necesario comprobar el final de cada línea, puesto que las palabras que exceden la capacidad de la línea son transferidas automáticamente al principio de la siguiente;
- posibilidad de inserción de caracteres, palabras o líneas, dentro del texto ya existente, para realizar correcciones o añadidos;
- borrado de caracteres, palabras y frases con la eliminación automática del espacio nuevamente disponible en cada línea;
- posibilidad de puesta en página automática, ya porque se desee variar el número de caracteres por línea, ya porque se varie el espaciado entre líneas;
- posibilidad de búsqueda automática de una palabra o un grupo de palabras dentro del texto;
- posibilidad de copiar y transferir partes de



texto a otro lugar del mismo documento.
En programas más evolucionados existen otras funciones más específicas que las indicadas, pero no por ello de menor utilidad: el principio general de funcionamiento sigue siendo el mismo. Como es natural, cualquier tratamiento de textos permite guardar el texto sobre un soporte magnético.

Hojas de cálculo

Una de las aplicaciones con mayor fortuna y difusión en el sector de ordenadores domésticos y personales, es con seguridad el tratamiento de hojas tipo tabla (aquellas que, para entendernos, están organizadas en líneas y columnas), es decir, la llamada hoja de cálculo electrónico o «worksheet» (y también «spreadsheet»).
El uso práctico de estos sistemas es muy amplio y variado; desde las simulaciones de ventas en función de los niveles de producción, hasta la gestión de balances personales o familiares, o el examen de la marcha de los costes, ventas y beneficios de una actividad comercial durante el transcurso de los doce meses del año.
Veamos ahora como funcionan.
Se trata de un conjunto de líneas y columnas localizables mediante un código (por ejemplo, letras para las columnas y números para las líneas, igual que en el juego de los barcos). Cada coordenada puede contener un dato numérico, una fórmula que conecte y elabore, algebraica o lógicamente, más datos, o un conjunto cualquiera de caracteres, como los componentes de un título. Entre las distintas informaciones que pueden aparecer en dos o más casillas se pueden definir relaciones analíticas, algebraicas o de otro tipo (por ejemplo, estadístico).
Una vez hecho ésto es suficiente con introducir mediante el teclado los distintos valores de las casillas definidas como variables principales, para que todas las variables dependientes sean calculadas automáticamente por el programa y vayan apareciendo en las casillas pertinentes.
De esta característica deriva la capacidad de las hojas de cálculo para ser empleadas como potentes instrumentos de «simulación» para modelos de tipo estadístico o financiero.
La compra de una hoja de cálculo electrónica resulta aconsejable para todos aquellos


ENERO
FEBRERO
MARZO
ABRIL
MAYO
JUNIO
CASA
GAS
LUZ
VIAJES
LETRAS
TOTAL

casos en los que exista una exigencia razonablemente frecuente de tener que elaborar cuadros de datos correlacionados entre sí.

Base de datos

Las bases de datos, o, en traducción libre, programas para la organización de datos, son programas de empleo generalizado aptos para encontrar y memorizar datos y noticias, archivar y clasificar impresos o conjuntos de informaciones, y así sucesivamente.
Una base de datos permite manejar electrónicamente cualquier tipo de archivo de informaciones, dispuesto y estructurado según los deseos del usuario. Un ejemplo lo aclarará fácilmente.
Supongamos que deseemos componer una agenda telefónica electrónica, que contenga, además del nombre y el apellido y el número telefónico de varias personas, también la dirección y la ciudad de residencia. Con una base de datos después de haber introducido los elementos es posible ordenar la lista de las personas por ejemplo en orden alfabético, o bien por zonas de residencia, por prefijo telefónico, o por cualquier otra «clave». Además se puede buscar inmediatamente de manera sencilla y automática, «la persona que tiene el prefijo X y que vive en la ciudad Y».
Volviendo sobre aspectos generales, las aplicaciones de una base de datos son prácticamente infinitas: tantas como los usos de archivos de información. Lo importante es que estos programas permiten el manejo de cualquier tipo de información, no importa cómo sea de rara y complicada, buscándola y poniéndola al día con la máxima rapidez y automatización.



Código máquina: los registros

El microprocesador montado sobre el Spectrum (pero en general, todas las CPU) dispone de un determinado número de «localizaciones» especiales llamadas registros, que para el programador en código máquina pueden ser comparables a las variables del programador en BASIC. En el Z80 existen aproximadamente 10 registros, cada uno de los cuales es conocido mediante una letra del alfabeto; de ellos, los principales son:

A, B, C, D, E, H, L

Cada registro puede ser considerado como una localización especial de la memoria, que, en consecuencia, puede contener un valor numérico comprendido entre 0 y 255. Sin embargo, existe (y esta es una de las ventajas del Z80) la posibilidad de agrupar por parejas los registros HL, BC y DE, de forma que se puede trabajar con valores numéricos superiores a 255. Naturalmente este hecho no impide en ningún caso el uso independiente de los registros.
Examinaremos ahora con mayor atención alguno de los registros del Z80, hablando un poco de las funciones envueltas en su uso, porque algunos de ellos, como veremos en breve, están especializados, y no es posible utilizarlos para cualquier circunstancia.
– El registro A. Normalmente denominado «acumulador», es el registro más importante del microprocesador. El es quien «acumula» la mayor parte de las tareas de funcionamiento de la CPU. Como compensación acepta, contrariamente a los demás registros, todos los modos de direccionamiento disponibles (los modos de direccionamiento te serán presentados en breve). Todas las oraciones aritméticas, lógicas y de comparación lo usan como intermediario.
– El registro B. Este registro sería bastante trivial si no se empleara de una forma especial en una instrucción de salto. Se puede asociar con el registro C para formar un registro doble.
– Los registros C, D, E, H, L. Son simples registros de 8 bits que se pueden usar por parejas: BC, DE, HL. Se puede acceder muy rápidamente a ellos y generalmente se usan para conservar los datos.
Cada uno de ellos es privilegiado con respecto a determinadas operaciones. Por ejemplo, el registro C se usa en las operaciones de Entrada/Salida. Los registros D, E, H, L se emplean en cambio para la transferencia de datos.
– El registro F. Este registro es el «abanderado» del microprocesador, en el

sentido de que cada uno de sus bits indica el tipo de resultado consiguiente a una determinada operación. Los bits del registro de estado (numerados del 0 al 7, siete el de la izquierda y 0 el de la derecha) se especifican con nombres concretos que ahora examinaremos brevemente.
.C, FLAG de carry, posición 0, contiene el bit más significativo de la última operación realizada. Es decir, el bit de carry toma el resto generado, en el transcurso de una suma, de la suma de los dos bits más significativos.
.N, FLAG de la resta, posición 1, contiene 1 si la última operación ha sido una resta o un decremento.
.P/O, FLAG de Paridad/Overflow, posición 2, tiene un significado doble. En las operaciones aritméticas indica si ha existido un rebosamiento de la capacidad (Overflow), invediendo la posición más a la izquierda del byte, empleada para el signo. En las operaciones lógicas indica, en cambio, la paridad (se hace 1 cuando se verifica una paridad par).
.AC, FLAG de carry auxiliar, posición 4, se emplea en las operaciones ariméticas ejecutadas sobre números expresados mediante una codificación especial. Se trata de un «medio-carry», de aquí su nombre. Da las mismas indicaciones del bit de carry, pero sobre un valor de sólo 4 bits.
.Z, FLAG de cero, posición 6, está en 1 si la última operación ha tenido cero como resultado.
.S, FLAG del signo, posición 7, toma el valor del bit más significativo del resultado de la última operación lógica o aritmética realizada por el microprocesador.

Técnicas de direccionamiento

El Z80 no dispone de una gran variedad de instrucciones (en conjunto son 67). Desde este punto de vista otras CPU resultan claramente superiores, llegando fácilmente a tener un número de instrucciones casi doble.
A pesar de esto, el Z80 es uno de los microprocesadores más difundidos. La razón de ésto es bien sencilla: se debe a la amplia posibilidad de técnicas de direccionamiento disponibles, lo que incrementa el número efectivo de instrucciones a más de 600.
Veamos en primer lugar que es lo que entendemos con el nombre de «técnicas de direccionamiento».
Ya hemos comentado que el Asemmbler es un lenguaje de bajo nivel, escasamente (o mejor dicho, nada) estructurado y dotado

de formas lógicas verdaderamente mínimas. En otras palabras, el Assembler se vale de instrucciones de una extremada sencillez operativa y ejecutiva: todo se basa en números y localizaciones de memoria.
Y precisamente, es en virtud de este hecho, y para poder trabajar lo mejor posible en cualquier parte de la memoria, por lo que los constructores de los microprocesadores intentan introducir la máxima flexibilidad en las operaciones ejecutables, proporcionándole a la CPU la mayor cantidad posible de técnicas de direccionamiento
El direccionamiento hace referencia al modo en que debe ser considerado el operando en una determinada instrucción, lo que influye en el resultado mismo de la operación.
Examinemos ahora, uno por uno, los distintos direccionamientos disponibles, evaluando sus posibilidades, sus usos, sus ventajas y sus desventajas.

Direccionamiento directo sobre un registro

El operando es un registro. Así:

```
LD A,B
```

significa «carga en A el contenido de B». Las instrucciones de direccionamiento sobre un registro ocupan únicamente un byte; en consecuencia, no solamente son breves, sino que permiten una alta velocidad. El tiempo necesario para su ejecución se limita a sólo 4 ciclos de reloj, que en el Spectrum corresponden a menos de una millonésima de segundo.
Nota: por ciclo de reloj se entiende el tiempo que la CPU necesita para realizar una operación elemental. Este tiempo es medido inflexiblemente por un oscilador (una especie de reloj de cuarzo en miniatura). Ten en cuenta, a título de información, que el Z80 ejecuta más de 4 millones de operaciones elementales en un segundo, y ésto durante todo el tiempo que lo mantengas encendido.

Direccionamiento implícito

El código operativo «implica» la dirección del operando o de uno de los operandos: RRA sirve para ejecutar una operación hacia la derecha de un bit del registro A. Bit 0,A examina en cambio el bit de posición 0 del registro A.

Direccionamiento inmediato

Esta forma de direccionamiento se utiliza para cargar el acumulador o cualquier otro registro con un valor específico.
Por lo tanto, todas las instrucciones en el modo de direccionamiento inmediato tienen una longitud de dos bytes. El primero contiene el código operativo y el segundo contiene la constante numérica a cargar en el registro o a utilizar para una instrucción aritmética o lógica. Por ejemplo, si nosotros quisiéramos

cargar en el acumulador el valor 160 (A0) hexadecimal en modo inmediato, podríamos escribir:

```
LD A, A0H
```

Esta instrucción dice: «carga en el acumulador el valor hexadecimal A0».

Direccionamiento directo

El direccionamiento directo es el modo en el cual los datos son normalmente recuperados desde la memoria. Está especificado por un código operativo, seguido por una dirección de 16 bits encerrada entre paréntesis.
El direccionamiento directo necesita por lo tanto tres bytes. Un ejemplo de direccionamiento directo es:

```
LD A, (1234H)
```

Esta instrucción especifica que el contenido de la localización de memoria número 1234 debe ser memorizado en el acumulador.
La desventaja del direccionamiento directo es que necesita instrucciones de tres bytes: se trata por lo tanto de un modo bastante lento.

Direccionamiento indirecto mediante registros

Para mejorar la eficacia del direccionamiento directo se puede utilizar otro modo, el modo indirecto mediante registros.
Con este direccionamiento, una pareja de registros contiene la dirección en la cual se encuentra el valor a elaborar, es decir, el operando; la pareja de registros se indica entre paréntesis.
En la jerga informática se dice que los registros «apuntan» a la localización de memoria.
Un ejemplo de este modo es:

```
LD A, (HL)
```

que significa «carga en A el contenido del byte cuya dirección se encuentra en HL».
El direccionamiento indirecto mediante registros es notablemente más veloz que el indirecto, ya que la CPU no debe leer la dirección de memoria, que se encuentra ya en los registros.
Naturalmente, es necesario cargar en la pareja de registros la dirección deseada, y por tanto este método resulta solamente ventajoso cuando la dirección se va a utilizar más de una vez.

Direccionamiento relativo

Este método de direccionamiento es bastante utilizado para las instrucciones de salto (es decir, para los comandos que transfieren la ejecución de un punto a otro del programa).
Por definición, el direccionamiento relativo utiliza dos bytes: el primero es una instrucción de salto, mientras que el segundo especifica el desplazamiento y su signo. ¿Qué significa ésto? Por ejemplo, supongamos querer mandar la ejecucion de un cierto punto del programa a otro distinto.
Para hacer ésto debemos especificar (además, naturalmente, de las instrucciones de salto) el número de localizaciones que queremos saltarnos. Ya que el desplazamiento pueder ser positivo o negativo (es decir, puede suceder tanto hacia adelante como hacia atrás), la instrucción de salto —que debe afectar como máximo a 255 localizaciones— puede ser negativa como máximo hasta 128, o bien positiva hasta 127 (salto hacia adelante de 127 localizaciones).
Naturalmente, la mayor desventaja de este modo es que no permite saltos superiores a 128 localizaciones.
Sin embargo, ya que la mayor parte de los bucles tiende a ser breve, este tipo de desviación puede ser utilizado casi siempre y ayuda a resolver el problema.

Direccionamiento indexado

El direccionamiento indexado es una técnica especialmente útil para acceder a los datos contenidos en un grupo de localizaciones, uno detrás de otro. El principio consiste en que la dirección deseada se calcula sumando a la dirección contenida en uno de los dos registros índice (los registros índice IX e IY son otras dos parejas de registros del Z80) el valor del operando.
Así

```
LD E, (IX+5)
```

carga en el registro E el contenido del byte que contenga la dirección formada sumando el número 5 al contenido de la pareja de registros IX.
Todos los modos de direccionamiento (para simplificar hemos omitido algunos ¿está bien?) requieren en cualquier caso, para ser adecuadamente comprendidos y utilizados de la mejor forma, bastante tiempo y (sobre todo) mucho trabajo de programación. Los ejemplos que veremos dentro de poco ilustrarán algunas posibles aplicaciones de estas técnicas.

Incremento/ decremento

La operación aritmética más sencilla que el Z80 es capaz de ejecutar es la operación algebraica de adición y sustracción del valor 1 al valor contenido en un registro. A primera vista, esta posibilidad no parece abrir grandes horizontes: en realidad, la disponibilidad de una operación de este tipo permite la construcción de estructuras muy complejas, como por ejemplo los bucles FOR... NEXT del BASIC.
La importancia de las instrucciones de adición y sustracción es tan grande que existen dos instrucciones específicas para llevar estas misiones a término.
De estas instrucciones existen dos formas distintas, disponibles con diferentes modos de direccionamiento:

ADD	instrucción de suma sin resto
ADC	instrucción de suma con resto
SUB	instrucción de resta sin resto
SBC	instrucción de resta con resto

Todas estas instrucciones pueden influenciar a algunos bits del registro de estado, concretamente, el bit de signo negativo y el bit de cero.
El bit del signo se pone en 1 si el bit más significativo del registro (es decir, el que ocupa la posición 7) se pone a 1 a continuación del incremento o decremento; de otra forma vale cero.
El FLAG de cero se pone a 1, en cambio, solamente si, a continuación de una ejecución, el registro que se ha usado contiene el valor cero.
Por ejemplo, incrementando en 1 el valor FFH (255 hexadecimal) contenido en un determinado registro, se llevará a 00H el valor del registro, se pondrá a 1 el FLAG de cero (Z=1) y se anulará el FLAG de signo negativo. Por otro lado, decrementando un registro que contenga 00H, se llevará el valor del registro a FFH, poniendo simultáneamente a 1 el FLAG de señal y a cero el FLAG de cero.

LENGUAJE

Los bucles

Los bucles son una parte importantísima —aún más, vital— en cualquier lenguaje de programación: en Assembler lo son todavía más, ya que muchas instrucciones, que en lenguaje de alto nivel necesitan de un único comando, para su ejecución, necesitan en cambio secuencias y repeticiones más o menos largas en código máquina.

Antes de pasar a la demostración práctica del uso de los bucles en Assembler es necesario, sin embargo, persentar otros dos grupos de instrucciones: las de comparación y las de salto.

Las comparaciones

Es posible comparar el valor contenido en el acumulador del Z80 con el contenido de una localización de la memoria (o bien con un valor numérico cualquiera) mediante las instrucciones

CP	compara el valor
CPI	compara el valor con incremento
CPD	compara el valor con decremento

El resultado de la comparación altera el FLAG de cero (Z), de signo (N) y de resto o carry (C) del registro de estado. ¿De qué modo? La respuesta está en esta pequeña tabla:

ACUMULADOR	CARRY	CERO	SIGNO
menor que el dato	1	0	1
igual que el dato	0	1	1
mayor que el dato	0	0	1

Comparando, por ejemplo, el valor del acumulador (supongamos que sea 5AH) con el número B3H, obtenemos de la tabla que el valor de las diversas FLAG será:

C=1 Z=0 N=1

Sobre la base de estos resultados podremos entonces tomar las decisiones más apropiadas.

Los saltos

Una vez efectuada la operación de comparación puede ser necesario ejecutar también una operación de salto (así como en BASIC escribimos por ejemplo IF A > B THEN GO TO 300).
Podemos distinguir tres grupos de saltos:
- los saltos propiamente dichos
- las llamadas a los subprogramas
- el retorno desde los subprogramas.

También hay dos modalidades de saltos:
- absolutos
- relativos.

Una instrucción de este tipo:

```
JP ADR
```

se llama salto incondicional; provoca el salto sistemático a la dirección ADR (indicando con ADR una dirección genérica de la memoria).
Es el equivalente en código máquina al GO TO que se usa en BASIC. Estas otras instrucciones son, en cambio, ejemplos de saltos condicionales:

```
JP C,ADR        JP NC,ADR
JP Z,ADR        JP NZ,ADR
JP M,ADR        JP P,ADR
JP PE,ADR       JP PO,ADR
```

«Salto condicional» significa que el salto debe ser efectuado solamente si la condición requerida ha sido satisfecha.
Naturalmente, las condiciones de salto condicional se pueden presentar en un programa solamente detrás de instrucciones que posicionen de cualquier manera las FLAG. La condición de salto es comprobada en base a los valores asumidos por cualquiera de estas FLAG.

```
....
CP 8            ; compara A con 8
JP Z, ADR1      ; salta si A=8
JP C, ADR2      ; salta si A<8
....            ; entonces A>8
```

En este ejemplo, la instrucción CP

posiciona los flag Z y C. El salto condicional JPZ comprueba la presencia del flag Z y provoca el salto a la dirección ADR1, si Z está a 1, señalando de esta manera que el contenido del acumulador es igual a 8. En caso contrario el programa prosigue y encuentra la instrucción JPC, que comprueba el estado del flag C. Si éste está a 1 se efectúa un salto a la dirección ADR2; de esta forma podemos estar seguros de que el contenido de A es mayor que 8, ya que no satisface el test precedente, y el programa continúa secuencialmente.
Hasta ahora hemos visto como operando de la instrucción JP a una instrucción de salto. Sin embargo, es posible efectuar un salto no solamente a una dirección dada, sino también a una dirección calculada, contenida en uno de los registros HL, IX o IY.

```
JP (HL)
JP (IX)
JP (IY)
```

Esta modalidad de salto se hace muy útil en muchas ocasiones.
Los comandos de SALTO RELATIVO se distinguen de los demás por una característica especial: especifican un desplazamiento relativo, es decir, hacia adelante o hacia atrás de +127 localizaciones o de −128 bytes, respecto a la posición ocupada en ese momento. Las instrucciones de salto relativo son:

```
JR       VAL
JR C,    VAL      JR NC, VAL
JR Z,    VAL      JR NZ, VAL
```

VAL representa un valor numérico que indica —en modo relativo— la dirección a la cual saltar.
Por ejemplo:

```
JR Z, -14
```

significa: «si el resultado de comparación (que se habrá realizado ya) ha puesto el flag Z a 1, entonces ejecuta un salto hacia atrás de 14 bytes».
Además de las instrucciones propiamente de salto, existe una última instrucción de salto: CALL.
El CALL del Assembler es equivalente al GOSUB de BASIC.
Esta instrucción efectúa un salto al subprograma cuya dirección está especificada en el operando, no sin haber efectuado antes una grabación en el área de stack de la dirección de vuelta. Este último punto es el que diferencia CALL de una simple JP, a pesar de lo cual, el parecido entre ambas es muy notable.
Para poder volver de los subprogramas en código máquina, igual que en BASIC es

necesario recurrir al RETURN, es necesario, en cambio impartir una instrucción RET. En este caso el microprocesador extrae un valor del área de stack y devuelve la ejecución del programa a la dirección correspondiente al valor tomado.
Terminada finalmente la parte teórica pasaremos ahora a la práctica, es decir, a las aplicaciones de los conceptos que hemos visto hasta ahora.
Comenzamos con un ejemplo fácil, fácil...: supongamos que queremos escribir un carácter (por ejemplo una «A») en el ángulo superior izquierdo de la pantalla. Sabemos ya cómo hacerlo en BASIC: basta un PRINT AT y el juego está hecho. Pero ahora debemos olvidarnos de estas comodidades y tratar de hacerlo todo con nuestros medios. Antes de nada es necesario recordar el mecanismo por el cual los caracteres pueden aparecer en la pantalla: cada posición de la pantalla está compuesta por una «red» de 8 x 8 cuadraditos. Cada grupo de 8 cuadraditos dispuestos sobre la misma fila corresponden a un byte; cada carácter se escribe cumplidamente con 8 bytes. Para visualizar en pantalla un cierto carácter bastará, por tanto, memorizar en las 8 localizaciones correspondientes a la posición elegida, los 8 valores que lo identifican. Dado que la memoria de pantalla comienza en la localización 16384, para visualizar nuestra A en BASIC podremos hacer

```
POKE 16384,0
POKE 16640,60
POKE 16896,66
POKE 17152,66
POKE 17408,126
POKE 17664,66
POKE 17920,66
POKE 18176,0
```

Ejecutando estas instrucciones en modo inmediato, veremos aparecer, un trocito cada vez, nuestro carácter A. Examinemos ahora cómo hacerlo en código máquina. El procedimiento no cambiará mucho: será necesario cargar en memoria los diversos valores:

```
LD A,0
LD (4000H),A
LD A,60
LD (4100H),A
LD A,66
LD (4200H),A
LD (4300H),A
LD A,126
LD (4400H),A
LD A,66
LD (4500H),A
LD (4600H),A
LD A,0
LD (4700H),A
```

Cargamos el código máquina utilizando el habitual progama BASIC:

```
10 CLS:RESTORE:CLEAR 32500
20 LET PRINCIPIO=32500:LET FIN=33000
30 FOR X=PRINCIPIO TO FIN
40 READ A
50 IF A=999 THEN GO TO 80
60 POKE X,A
70 NEXT X
80 RANDOMIZE USR 32500: STOP
90 DATA...aquí se escriben los códigos...
```

Para insertar los

códigos del programa en código máquina en la línea DATA debemos ejecutar la conversión de las instrucciones mnemónicas en los correspondientes valores numéricos:

ASSEMBLER	CODIGOS HEXADECIMALES	CODIGOS DECIMALES
LD A,0H	3E,0	62,0
LD (4000H),A	32,00,40	50,0,64
LD A,3CH	3E,3C	62,60
LD (4100H)	32,00,41	50,0,65
LD A,42H	3E,42	62,66
LD (4200H),A	32,00,42	50,0,66
LD (4300H),A	32,00,43	50,0,67
LD A,7EH	3E,7E	62,126
LD (4400H),A	32,00,44	50,0,68
LD A,42H	3E,42	62,66
LD (4500H),A	32,00,45	50,0,69
LD (4600H),A	32,00,46	50,0,70
LD A,0H	3E,0	62,0
LD (4700H),A	32,00,47	50,0,71

Recordando además que nuestro programa BASIC, para no verse interrumpido por un mensaje de error, necesita que el último valor sea el número 999.

Ejecutando el programa, verás aparecer sobre la pantalla, como antes, la letra «A», en la posición correspondiente a las localizaciones afectadas por la rutina en código máquina.

El ejemplo que acabamos de ver es bastante interesante pero no resulta muy práctico: la longitud de la rutina en Assembler parece excesiva para

una tarea así de sencilla.
¿Por qué especificar la forma del carácter «A», dado que en la propia memoria ROM ya sabe el Spectrum todas sus características?
Intentaremos pues recurrir a una segunda solución. Para imprimir el carácter en pantalla deberemos primero saber dos cosas:
1) la dirección donde están memorizados los 8 números propios del carácter que deseamos imprimir
2) la dirección (o mejor, las direcciones) que corresponden a una determinada posición de la pantalla.
En el caso de la letra A, y de la primera posición en la esquina superior izquierda, estas direcciones son, respectivamente, la 15880 y la 16384.
Para imprimir el carácter bastará sencillamente con tomar 8 veces un byte de la «memoria de caracteres» y depositarlo en una determinada localización de la «memoria de pantalla».
He aquí el aspecto de una posible solución:

```
LD HL,3E08H
LD BC,4000H
LD A,8H
PUSH AF
LD A,(HL)
LD (BC),A
INC HL
INC B
POP AF
SUB 01
JR NZ, −10
RET
```

El programa empieza memorizando en las dos parejas de registros HL y BC las direcciones de la memoria de caracteres y de la memoria de pantalla.
El acumulador, además, es cargado con el valor 8 (LD A,8A).
Aquí es donde empieza el bucle propiamente dicho.
1) PUSH AF sirve para memorizar en una zona segura del microprocesador (llamada área del STACK) el valor del acumulador.
2) Se carga el registro A con el valor contenido en la dirección HL.
La instrucción LD,A(HL) es un ejemplo de direccionamiento indirecto mediante registros.
3) Se memoriza A en la localización a la que apuntan los registros BC.

```
LD (BC),A
```

4) Se incrementan las direcciones a las que apuntan HL y BC.
5) Se recupera (POP) del área de Stack el valor del acumulador anteriormente memorizado.
6) Si restándole 1 al acumulador todavía no se ha llegado a 0, entonces se vuelve a empezar el bucle. La instrucción JR NZ,−10 es un salto relativo concicional para terminar el bucle. El desvío se efectúa cuando los FLAG N y Z están en 1.
Como habrás podido observar esta rutina es mucho más breve que la anterior, y, en suma, bastante más elegante.
Los códigos numéricos que se obtienen por la conversión de las instrucciones en código mnemónico son:

```
33, 8, 62, 1, 0, 64, 62, 8, 245, 126, 2, 35, 4, 241, 214, 1, 32, 246, 201.
```

Herramientas de programación

Dado que cada vez nos introducimos más en las interioridades del Spectrum, podemos empezar a examinar algunos «Programas de utilidades», trucos y breves rutinas, que siempre serán de ayuda en multitud de circunstancias.
Veamos en primer lugar qué es lo que hay que hacer para imposibilitar el borrado de la primera línea de un programa; este truco, sin ninguna utilidad práctica aparente, les resultará muy útil a todos aquéllos que deseen que en la primera línea de sus programas aparezca una instrucción REM que contenga el nombre del autor o cualquier otro tipo de aviso.
El secreto es bien sencillo, será suficiente con impartir, después de haber establecido la línea, la siguiente instrucción en modo inmediato:

```
POKE 1+PEEK 23635+256*PEEK 23626,0
```

Esta técnica le asignará a la línea el número 0, de forma que ya no habrá modo de cambiarla o borrarla. El proceso que permite esta operación proviene del hecho de que en las localizaciones 23635 y 23636 se guarda la dirección del programa BASIC.
Con un mínimo de práctica (o, mejor dicho, de experiencia) descubrirás que cambiando el 0 de POKE por otro valor, se pueden lograr otros efectos, distintos al recién indicado, pero igualmente interesantes.
Veamos ahora un programa útil, que permite convertir un número cualquiera, escrito en una determinada base (A), en el mismo número con otra base (B).
Debido a la forma en que se ha planteado el programa, la máxima base que se puede manipular es 36. Las cifras superiores a 9 están representadas por letras del alfabeto (tanto en mayúsculas como en minúsculas, puesto que el programa no las distingue); así, por ejemplo: las doce cifras de un sistema en base 12 serán:

0 1 2 3 4 5 6 7 8 9 A B.

He aquí el listado:

```
 10 REM CONVERSION DE BASES
 20 INPUT "¿DESDE CUAL BASE?"; A
 30 INPUT "¿A CUAL BASE?"; B
 40 PRINT "DESDE LA BASE:";A,"A LA BASE:";B
 50 INPUT "¿QUE NUMERO DESEAS CONVERTIR?"
    LINE A$
200 REM DE BASE A A BASE 10
210 LET S=0
220 FOR C=LEN(A$) TO 1 STEP -1
230 LET D=CODE(A$(C))-48-(7 AND A$(C)>
    ="A")-(32 AND A$(C)>="a")
240 LET S=S+D*A↑(LEN A$-C)
250 NEXT C
300 REM DE BASE 10  A BASE B
310 LET B$="":PRINT
320 LET R=INT (S-B*INT(S/B)+0.5):LET S=INT(S/B)
330 LET B$=CHR$(R+48+(7 AND R>9))+B$
340 IF S< >0 THEN GOTO 320
350 PRINT A$;TAB 12;CHR$(61);TAB 16;B$;TAB 9
360 GO TO 10
```

El corazón del programa está en las líneas 230-240 y 320-330. Las expresiones matemáticas que puedes leer en estas instrucciones derivan de la manera en la que el Spectrum codifica los caracteres (de hecho, el número a convertir es leído como cadena de caracteres, no como un número). Finalmente, podrás observar que el paso de la base A a la base B se efectúa mediante la transformación intermedia del número de la base A a la base 10.

Examinemos ahora otro programa, de tipo completamente diferente al anterior, pero no por ello menos útil: se trata de una rutina de renumeración. Como ya sabrás, un programa de renumeración es de gran utilidad para la programación, para poder reordenar automáticamente todos los números de línea en el momento en que empiecen a ofrecernos problemas por su confusión.
La versión que proponemos es de tipo «aficionado», en el sentido de que la reordenación se efectúa excluyendo los GOTO y los GOSUB (que, por lo tanto, habrá que modificar manualmente).

Por otra parte, si hubiéramos incluido la renumeración de los saltos, el programa se habría hecho mucho más largo y complicado.

```
9000 REM RENUMBER
9010 STOP
9020 INPUT "INDICA EL NUMERO DE LA
     PRIMERA LINEA A RENUMERAR",,NIR
9030 INPUT "INDICA EL NUEVO NUMERO DE
     LINEA INICIAL",,NLI
9040 INPUT "INDICA EL INTERVALO ENTRE
     LINEAS",,ITL
9050 LET IM=PEEK(23635)+256*PEEK(23636)-1
9060 LET NX=PEEK(IM+1)*256+PEEK(IM+2)
9070 IF NX<NIR THEN GOTO 9120
9080 IF NX=9000 THEN GOTO 9140
9090 POKE (IM+1),INT(NLI/256)
9100 POKE (IM+2),NLI-256*INT(NLI/256)
9110 LET NLI=NLI+ITL
9120 LET IM=IM+4+PEEK(IM+3)+256*PEEK(IM+4)
9130 GO TO 9060
9140 LIST
```

El programa se tiene que poner en marcha con RUN 9020 o con GOTO 9020. Aparecerán los letreros de la entrada, que piden la indicación del número de línea desde el que tiene que empezar la renumeración (por ejemplo, 5) después del nuevo número de línea inicial (por ejemplo, 100) y finalmente, de la distancia, es decir, del paso entre línea y línea. Naturalmente, también es posible efectuar más de una renumeración, por ejemplo, para tener las primeras 20 líneas renumeradas como 1000, 1050, 1100, etc. Por comodidad, el programa se puede memorizar en casete con la instrucción SAVE «RENUMBER», y puede ser cargado en memoria, cuando te sea necesario, con MERGE «RENUMBER», lo que permite insertar la rutina en la memoria sin que se borre el programa ya existente.
Para usar el RENUMBER el programa a renumerar no tiene que usar las líneas de la 9000 a la 9140; además hay que reservar para el funcionamiento de la rutina las variables IM, NIR, NLI, ITL, NX.
A título de curiosidad (pero no solamente) mostraremos ahora una versión de RENUMBER escrita en código máquina; ésta también desempeña la misma tarea que el programa en BASIC recién visto, es decir, se ocupa simplemente de renumerar las líneas

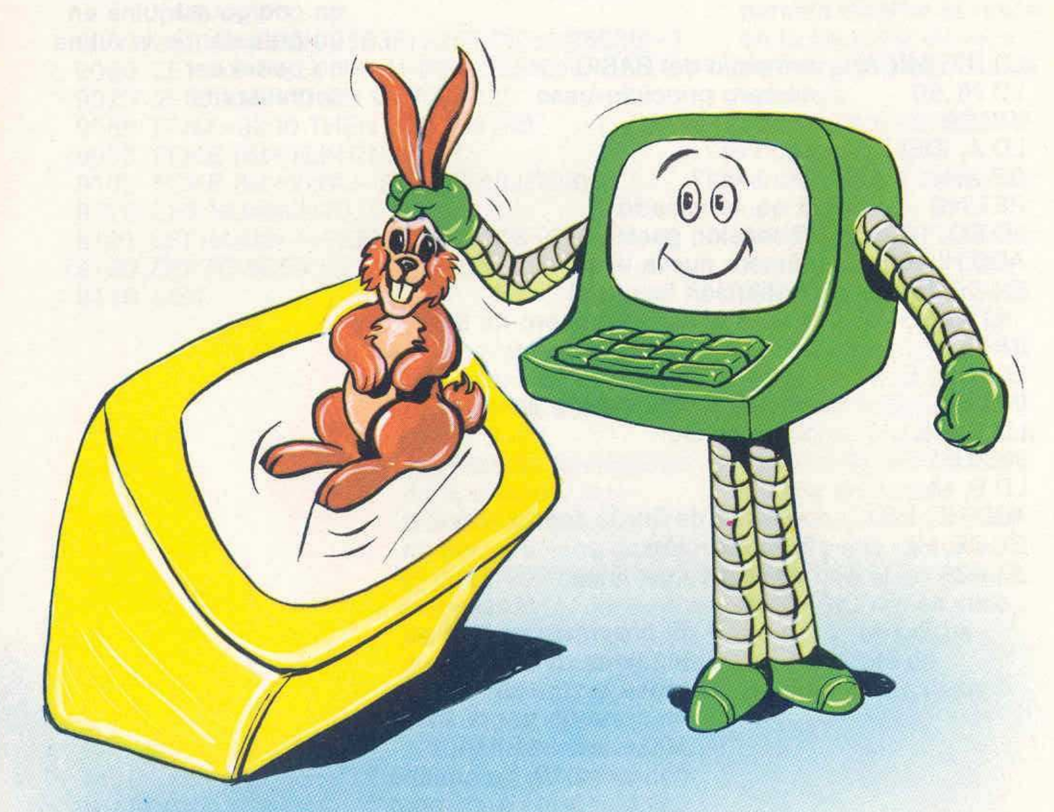
sin ocuparse de la posible presencia de GOTO, GOSUB y RESTORE. Sin embargo, lo espectacular es la extremada velocidad con la que se lleva a cabo la tarea; además, la versión en código máquina resulta mucho más corta que la correspondiente en BASIC.
He aquí el listado en Assembler:

```
LD DE, 5CCAH   ;principio del BASIC
LD HL,90       ;número principio-paso
INC DE
LD A, (DE)     ;¿qué es?
CP 28H         ;¿fin línea?
RET NC         ;si no, terminado
LD BC, 10      ;dimensión paso
ADD HL, BC     ;número nueva línea
EX DE, HL      ;sustitución temporal
  LD (HL), D   ;inserta el nuevo número de línea
INC HL
LD (HL), E
INC HL         ;toma la longitud de la línea
LD C, (HL)     ;ponla en BC
INC HL
LD B, HL
ADD HL, BC     ;posición de fin de línea
EX DE, HL      ;fin sustitución
JR –21         ;haz la próxima línea
```

El programa en código máquina renumera todas las líneas, empezando por la línea 100 e incrementando con paso 10. Es posible cambiar tanto el número de principio, modificando el argumento de la segunda línea (actualmente 90=100–10), como el incremento, sustituyendo con otro número el valor 10 de la instrucción LD BC, 10.
Para cargar la rutina en memoria te proponemos ahora un pequeño programa BASIC, que se ocupa de poner las instrucciones en código máquina en un área donde la rutina no podrá ser sobreescrita.

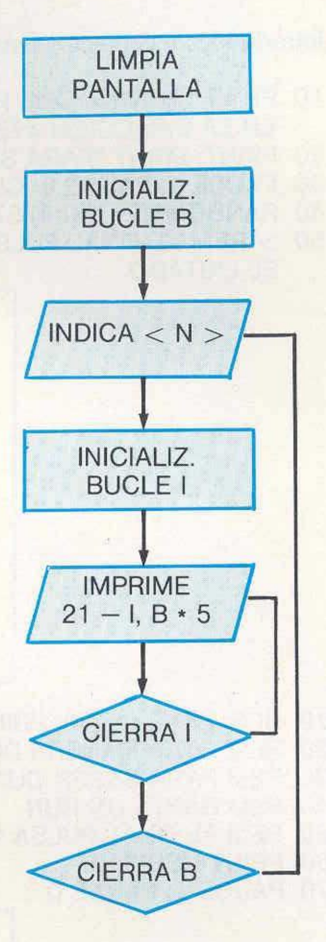
```
100 CLEAR 32499: LET A=32500
110 READ N: IF N=999 THEN STOP
120 POKE A,N
130 LET A=A+1
140 GO TO 110
150 DATA 17,202,92,33,90,0,19,26,254,40,208,1
160 DATA 10,0,9,235,114,35,115,35,78,35,70,9
170 DATA 235,24,235,999
```

Después de haber introducido el programa en memoria bastará con ejecutarlo para tener la rutina en memoria. Para hacer ejecutar la renumeración puedes impartir en modo inmediato una instrucción USR, como PRINT USR 32500 o LET B=USR 32500. ¡Acuérdate, los GOTO y los GOSUB no se renumeran!



Representación gráfica

He aquí un programa para generar un diagrama de barras. La pantalla visualiza hasta 5 elementos verticales capaces de representar valores comprendidos entre 0 y 21. El carácter utilizado para imprimir las barras se obtiene pulsando CAPS SHIFT y 8 en modo gráfico (cursor G).
Observa el anidado de los bucles automáticos, fácilmente observables en el diagrama de flujo.




```
10 CLS
20 FOR B = 1 TO 5
30 INPUT "N =";N
40 FOR I = 0 TO N
50 PRINT AT 21 - I, B * 5; "███"
60 NEXT I
70 NEXT B
```

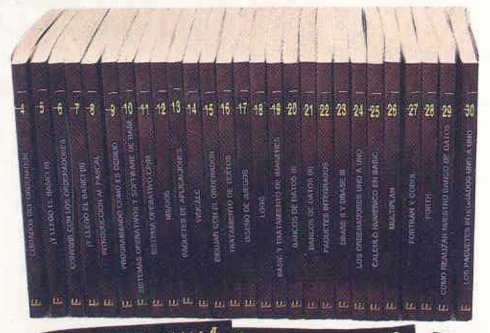
EJERCICIOS

Introduce los siguientes listados y atente a lo que se dice en las instrucciones REM.

```
10 PRINT "ADIVINA QUE HACE LA RUTINA DE LA ROM SITUADA
   EN LA DIRECCION 4757"
20 PRINT: PRINT "PARA SABERLO PULSA UNA TECLA"
30 PAUSE 1: PAUSE 0: CLS
40 RANDOMIZE USR 4757
50 > REM AL FINAL PULSA LIST PARA COMPROBAR SI EXISTE TODAVIA
   EL LISTADO
```

```
10 CLS: BEEP 0.5,30: PRINT "PULSA UNA TECLA": PAUSE 1: PAUSE 0
20 REM OTRA MANERA DE EFECTUAR UNA INSTRUCCION
30 REM PARA SABER CUAL SOLO TIENES QUE PROBARLO
40 REM BASTA UN RUN
50 REM AL FINAL PULSA UNA TECLA
60 PRINT USR 6137
70 PAUSE 1: PAUSE 0
```

UNA GRAN OBRA A SU ALCANCE



UNA OBRA COMPLETISIMA EN 30 VOLUMENES QUE TRATA TODOS LOS TEMAS , DESDE QUE ES UN ORDENADOR HASTA EL ESTUDIO DE LOS DIVERSOS LENGUAJES, PASANDO POR LOS LENGUAJES, METODOS DE PROGRAMACION, ELECCION DEL ORDENADOR ADECUADO, DICCIONARIO, ETC.



B.B.I.
INGELEK

30 EXTRAORDINARIOS VOLUMENES DE APARICION SEMANAL CON TODOS LOS CONCEPTOS DE LA INFORMATICA

GRAN OFERTA DE SUSCRIPCION
10.800 PTAS.



OFERTA VALIDA UNICAMENTE PARA ESPAÑA

AHORRE MAS DE 1.000 PTAS Y LLEVESE UNA MAGNIFICA CALCULADORA SOLAR VALORADA EN 2.500 PTAS.

La publicación de este anuncio anula los precios anteriores. Oferta válida hasta el 28-2-86.

SUSCRIBASE POR TELEFONO

Todos los días, excepto sábados y festivos, de 8 a 6,30 atenderemos sus consultas en el



2505820